seq2graph: A Neural Approach to Scene Graph Generation from Natural Language

Frits van Paasschen
Department of Computer Science
Stanford University
Stanford, CA 94305
`fritsvp@cs.stanford.edu`

Isaac Kasevich
Department of Computer Science
Stanford University
Stanford, CA 94305
`isaack97@cs.stanford.edu`

Abstract

In recent years, research has explored the possibility of encoding image semantics in graph-based representations that capture both high and low-level features of the images they describe. These encodings, generally known as 'scene graphs' [13], can be used in a variety of applications from image search and retrieval to image generation [8] [10]. Given the usefulness of these scene graphs, recent research has explored the possibility of not only creating scene graphs from images [2], but also of generating scene graphs from a natural language description of a scene [1]. This is especially poignant in applications related to image retrieval and scene generation [8], as natural language descriptions present an obstacle in achieving consistency and accuracy in scene generation and image retrieval systems. In this paper, we present our approach towards a neural end-to-end system for parsing paragraph level natural language descriptions of images into scene graphs. Overall, though we still believe our approach could work, we were unable to successfully build and test a fully viable end-to-end neural solution to this problem.

1 Introduction

A scene graph, defined by [8] and [13], is a graph $G = \{V, E, A\}$ that defines the semantic structure of a scene or image such that $V = \{v_1, ..., v_2\}$ is a multiset of the objects in the image, $E = \{(v_i, e_k, v_j) | v_i, v_j \in N, e_k \in R\}$ is a set of relationships (R being the set of all relationship types and N being the set of all node types) between these objects, and $A = \{(v_i, a_k) | v_i \in V, a_k \in Q\}$ is a set of attributes connected to each object in the image (Q being the set of all attribute types). The goal of this project is to provide a flexible neural solution to generate semantically rich scene graphs from natural language descriptions, not the images they describe.

Scene graphs have a variety of applicable use-cases. For example, they have been used to generate the scenes they describe [10], and for more accurate image retrieval within image databases [1] [8]. As such, a system that could reliably convert paragraph-level descriptions of scenes into their corresponding scene graphs could be utilized in a variety of applications. We motivate our project on this main principle.

2 Related Work

Although we take a novel approach to scene graph generation, the general strategy behind a significant amount of our approach closely resembles that proposed in [1]. We build on this approach using techniques defined in [24], [2], [18], and [5]. Similar work that we experimented with is also proposed in [20], [27], [11], [12], [15], and [16].

Perhaps the most applicable work in this field of research to our project of scene graph generation from natural text has come from [1], where scene graphs are generated from one-sentence descriptions using the following pipeline: first, the sentences are parsed into grammatical dependency graphs in a form consistent with [7] and [26]. Then, these dependency trees are modified by accounting for quantifiers, using co-reference resolution [14], and by accounting for plural nouns. From there, the authors of the paper use tree-regexes [28] and a classification model to extract object node types, to predict relationships, and to extract object attributes. Evaluated on a downstream metric of image retrieval, this approach succeeds at capturing the semantics of a sentence and representing these same semantics in a scene graph. While this approach did successfully generate scene graphs from sentences, it predicts nodes and edges independently. Further research in graph generation [11] has shown that this type of approach is suboptimal.

Recurrent neural networks have been applied to scene graph generation given input images or data [18] [2] [11]. By modeling the generation of a graph as a sequence of node and edge adding actions, recurrent neural networks are able to model complex graph dependencies and generate semantically viable graphs. Additional research into graph generation has generally used paired RNNs in the form of GRUs or LSTMs, one of which is used to sequentially add nodes to a graph, while the other is used to sequentially add relationships between added nodes and the rest of the graph [2] [11] [12]. While they have not yet been used to generate scene graphs from text, we view these model architectures as promising areas with which to experiment in designing a sequence-to-scene-graph model.

In our research and our approaches, we also considered research applications of attention [5], neural sequence models [2] [11], neural coreference resolution [23], neural dependency parsing [24], multitask learning [20] [27], and set prediction [16] [15].

Given existing research, we see an area of limited understanding into fully neural approaches towards generating scene graphs directly from textual descriptions. As such, we designed an end-to-end pipeline that does not rely on regular expressions or rich, hand-designed semantic rules and features as in [1], and that utilizes a novel combination of cutting edge techniques such as those proposed in [2], [24], [11], and [5].

3 Approaches

3.1 One-Shot Node Multiset Prediction

In the previous milestone, we began with the task of generating a node multiset from a paragraph-level input sequence.

We simply began with input paragraphs P of variable length and paired outputs in the form of multisets V. To represent these multisets computationally, we defined k-hot vectors, $v \in \mathbb{Z}^{|N|}$ (where $|N|$ denotes the total number of node types). In these vectors, nonzero indices indicate the presence of a certain nodes, and the scalar values at those indices indicate how many of each node type is present in the scene.

We represented the output as a *set* of object nodes rather than as a sequence of nodes, as there is no “ground-truth” ordering available due to the isomorphic structure of scene graphs. Additionally, [16] showed that we cannot simply choose an arbitrary ordering of nodes. Moreover, it is computationally intractable to maximize over all $|N|!$ possible orderings. While there has been substantial research into making this type of problem tractable [15] [16] [11], these are generally not applicable to multisets.

We experimented with multiple novel encoder-decoder pair architectures to attempt to solve the initial node-generation problem. All experiments use the same encoder architecture, which is shown on the left of Fig. 1.: We begin with a sequence of paragraph words, $[x_1, ..., x_n]$. We look up a word embedding $e_i \in \mathbb{R}^d$ for each word x_i. We then run the sequence of embeddings through a stacked, bidirectional LSTM with hidden and cell sizes $h_i, c_i \in \mathbb{R}^d$. After passing through the LSTM, we concatenate the first and last states hidden states to pass to the decoder;

$$dec = [\overleftarrow{h}_1; \vec{h}_n] \in \mathbb{R}^{2d}.$$

We experimented with two decoder architectures:

1. *Baseline: Dense Multi-Layer Perceptron (MLP) w/ ReLu*
   We pass the encoding output through multiple dense layers with ReLu activation. Notably, since the output must be of $\mathbb{R}^{|N|}$, we require the final linear layer to output a vector $\hat{y}$ of this

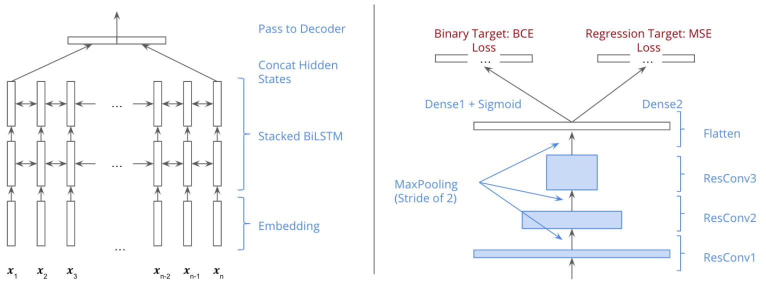

Figure 1: Convolutional Multitask Architecture

dimension. We then optimize with respect to Mean-Squared Error Loss commonly used in OLS regression:

$$\ell(y, \hat{y}) = \frac{1}{|N|}||y - \hat{y}||_2,$$

where y is the target $k-$hot vector.

2. *VGG-Inspired Multitask Convolutional Decoder with Max Pooling*
We first define the `ResConv` sub-block (commonly used in Computer Vision architectures) which consists of two successive $1d$ convolutions (with ReLu activation) and a skip (e.g. *residual*) connection from the input of the first convolution to the output of the second. We stack several of these `ResConv` blocks linearly, with MaxPooling layers between blocks to facilitate comprehension of higher-order features. We flatten the output of the final `ResConv` block and pass it through two separate dense layers. The output of the first dense layer is passed through a sigmoid activation, yielding a binary vector output $\hat{y}_b$. The output of the second dense layer is taken to be the multinomial vector output $\hat{y}_m$. We then generate a mask of the multinomial target vector y_m as a binary target y_b. The joint loss is then the weighted sum of an averaged Binary Cross-Entropy between y_m and $\hat{y}_m$ and the MSE loss discussed previously:

$$\begin{aligned}\ell(\hat{y}_b, y_b, \hat{y}_m, y_m) &= \lambda \ell_{BCE}(\hat{y}_b, y_b) + (1-\lambda)\ell_{MSE}(\hat{y}_m, y_m)\\ &= \frac{1}{|N|}\left[\lambda \sum_i \left(y_b^{[i]} \log(\hat{y}_b^{[i]}) + (1 - y_b^{[i]}) \log(1 - \hat{y}_b^{[i]})\right) + (1-\lambda)||y_m - \hat{y}_m||_2\right],\end{aligned}$$

where λ is the weighted-sum hyperparameter. This is an implementation of *multitask learning*, where we hypothesized that the model could learn to optimize for a difficult task (counting objects) by simultaneously optimizing for an easier task (recognizing the presence of an object). This architecture is shown on the right of Fig. 1.

Note: We also experimented with a fully-convolutional decoder (adding more and more output channels at each level until reaching $|N|$ in the final layer), though we abandoned this model after terrible initial performance.

3.2 Scene Graph Prediction using Dependency Parsing and Iterative Message Passing

Given the performance of this multitask architecture (discussed in the Experiments section), we decided to try a different novel approach that combines elements of both [1] and [2], which we call `seq2graph`. At a high level, we propose a candidate set of objects and edges (relations between objects) for each example, and use a variant of the model proposed by [2] to classify each object and relationship. Our proposed model proceeds as a pipeline over six core units, each of which will be further discussed in detail:

1. **Dependency Parser:**
A paragraph is split into raw input sentences, which are fed through an AllenNLP Dependency Parser [24], from which we extract candidate objects for each sentence.

2. **Candidate-Target Alignment [training only]:**
   Because scene graphs (i.e., a set of nodes and a set of relations between nodes) are inherently unordered, we must align target objects and relations between objects to candidates in an unsupervised manner (as in [1]) in order to train our downstream models.
3. **Feature Extraction:**
   Before attempting to classify candidate objects and edges, we must extract features from each:
   - **Object Feature:** the 100-dimensional GloVe [22] word vector for the candidate object.
   - **Edge Feature:** the projection of an attention-weighted concatenation of biLSTM hidden states run over the input sentence.
4. **Graph GRU:**
   We then feed node and edge features generated in (3) into a GraphGRU model as utilized by [2]. At a high level, each candidate object and edge maintains its own Gated Recurrent Unit (GRU) state as its representation. Instead of receiving messages from preceding time-steps as in a traditional RNN, messages are iteratively passed between nodes and the edges in which they participate.
5. **Softmax Classifiers:**
   We predict the node label for each candidate node and relationship label (if any) for each candidate edge with a standard softmax classifier with cross-entropy loss.
6. **Graph Merging:**
   Because we split the input paragraph i into n_i individual sentences to parse, we will produce n_i scene graphs that would need to be merged into a single, final scene graph. This is left for future work.

3.2.1 Dependency Parsing

For the first step in this ordered approach, we rely on a pretrained neural dependency tree parsing model [24] downloaded from [29]. To use this dependency parser, we first split an input paragraph into its sentence components. We then consider each sentence individually in the dependency parsing process and graph processing step. We note that while we do lose dependencies over multiple sentences (i.e., *"The man is running. The man is wearing a red hat."* should intuitively involve just one *man* object), we believe that we could successfully merge generated scene graphs for each sentence downstream. This dependency parser uses deep biaffine attention combined with a bidirectional LSTM over the input sequence to produce a dependency graph in a form consistent with Stanford Dependencies [26].

We then implement an algorithm with [25] in mind to extract salient node features from the dependency parse of a sentence. This algorithm uses a modified breadth first search of the graph to propose objects and numerical modifiers attached to each object to recreate the graph-processing step found in [1] (pictured in Fig. 2).

In addition to a modified BFS algorithm described in Algorithm 1, we attempted to incorporate neural coreference resolution using another pretrained model [23] from [29] for more complex sentences. For example, given the sentence *"There is a dog holding a stick, it is fluffy."*, we want our model to attribute the *fluffy* attribute to the *dog*, not the *stick* through coreference resolution on *it*. However, this model introduces significant complexity into our graph processing algorithm, and so for the scope of this project we chose to forgo using an external model for explicit coreference resolution.

3.2.2 Candidate-Target Alignment for Training

A critical component of our pipeline is candidate-target alignment. Because we only have access to full scene graphs and not direct pairs of objects or edges, we must create them in an unsupervised manner.

The task is defined as follows: given a set of identified candidate objects, a set of target objects, and a set of target subject-object-predicate relationship triples (target relationships), align (match 1-1) target objects to candidate objects. That is, we define a mapping $\tau : obj_{cand} \longrightarrow obj_{target}$ over all candidate and target objects in an example. We align objects in a hierarchical manner, first looking for direct matches between candidate and target, then looking for targets within increasing edit distances, and

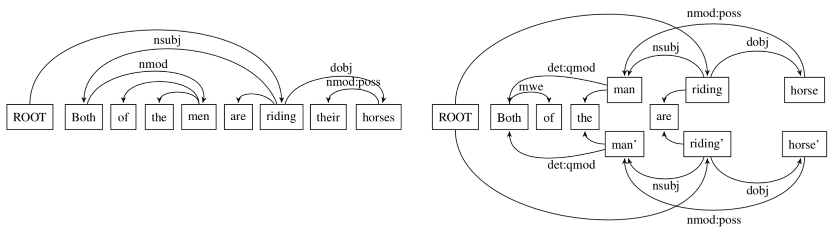

Figure 2: Graph Processing Step Replicated from [1]

Algorithm 1: Algorithm for Finding Raw Object Multiset from Parsed Dependency Graph

```
Result: Returns a list of objects given an input dependency graph
find_objects(root, objects, quantifiers);
if root is of subject/object type then
    search root subtree for other subject/object types using BFS;
    if subtree contains subj/obj types then
        add objects to list;
        delete current head from list;
    end
    use BFS to search subtree for numerical modifiers and add to quantifiers list;
    else
        find_objects(root.children, objects, quantifiers);
    end
    return objects, quantifiers;
end
```

finally taking the target with the closest word-vector cosine similarity. Then, we align relationships: we propose an edge between every candidate object pair (o_1, o_2). If $(\tau(o_1), \tau(o_2), \texttt{pred})$ exists in the target relationship set (for some predicate `pred`), we align `pred` as the target edge label for (o_1, o_2). If $(\tau(o_1), \tau(o_2), \_)$ *does not* exist, we align the special `<NONE>` relation to (o_1, o_2). We return set of (`candidate`, `aligned`) object pairs, and the set of (`candidate object pair`, `aligned relation`) edge pairs.

3.2.3 Edge Feature Extraction

To extract meaningful features for proposed edges, we first run a bi-directional LSTM over the input sentence. We then take the edge feature $f_{i\to j}$ as $\mathbf{w}_{edge}[\overleftarrow{h}_{\alpha(i)}; \vec{h}_{\alpha(j)}; \overleftarrow{h}_{\alpha(j)}; \vec{h}_{\alpha(j)}] \in \mathbb{R}^d$, where $\alpha(\cdot)$ is a function that finds the position index in a sentence where a word appears and $\mathbf{w}_{edge} \in \mathbb{R}^{4d\times d}$ is a learnable parameter. If a word appears more than once in a sentence, we simply take the average across those hidden states.

3.2.4 Graph GRU and Softmax Classification

Given a set of edge features and object features (simply their 100-dimensional GloVe word embeddings [22]), we then proceed with a new graph-decoder architecture proposed by [2] for scene-graph generation from images. For each node and edge, we maintain a Gated Recurrent Unit (GRU) cell. Instead of updating the cells sequentially, as in a normal RNN-architecture, we update them through iterative message passing. As described by [2], in the first iteration each node and edge state is randomly initialized and then fed, as input, the corresponding node or edge feature. In successive iterations, the states are updated with a message m, defined for nodes and edges respectively as:

$$\text{[node message]: } m_i = \sum_{j:i\to j} \sigma(\mathbf{w}_1^T[h_i; h_{i\to j}]) + \sum_{j:j\to i} \sigma(\mathbf{w}_2^T[h_i; h_{j\to i}])$$
$$\text{[edge message]: } m_{i\to j} = \sigma(\mathbf{v}_1^T[h_i; h_{i\to j}]) + \sigma(\mathbf{v}_2^T[h_i; h_{j\to i}])$$

Table 1: Dataset Statistics

# Examples	# Relationships	# Objects	Max Paragraph Length	Max # of Nodes
18,550	842	5099	304	167

where $\mathbf{w}_1, \mathbf{w}_2, \mathbf{v}_1,$ and $\mathbf{v}_2$ are learnable. After the last iteration, we run softmax classifiers over all NodeGRU states and EdgeGRU states to predict one label per node and one relationship (or `<NONE>`) per edge.

4 Experiments

4.1 Data

We relied on several sources to create a data set with mappings from descriptive paragraphs to scene graph representations of the images described by these paragraphs. We downloaded the latest version of the Visual Genome Scene Graph dataset [6]. Additionally, we used a dataset created by [19] that maps a subset of the Visual Genome set of images to descriptive paragraphs. By cross referencing the image identification numbers of the paragraphs from [19] and the scene graphs from [6], we were able to obtain a dataset of labeled pairs. See Table 1 for dataset statistics.

Below is a random (not cherry-picked) input/output pair example:

- **Paragraph:** *"A time clock hangs on the wall in the center of the image. A silver object is sitting on top of it. On either side of the clock hangs grey time card holders. Each slot is number. Below the time clock, on a shelf is a white hard hat with a black and grey chin strap. Under the helmet is a file folder.*
- **Object SynSet Multiset:** *{time_clock.n.01, wall.n.01, clock.n.01, glass.n.01, numeral.n.01, shelf.n.01, support.n.01, strap.n.01, paper.n.01, point.n.09, prison_guard.n.01, lock.n.01, mailbox.n.01, pipe.n.01}*
- **Relationship SynSet Multiset:** *{(time_clock.n.01, along.r.01, wall.n.01), (clock.n.01, be.v.01, glass.n.01), (numeral.n.01, along.r.01, panel.n.01), (support.n.01, have.v.01, shelf.n.01), (time_clock.n.01, have.v.01, point.n.09), (prison_guard.n.01, be.v.01, time_clock.n.01), (support.n.01, have.v.01, shelf.n.01), (time_clock.n.01, be.v.01, wall.n.01), (point.n.09, be.v.01, time_clock.n.01), (time_clock.n.01, have.v.01, lock.n.01), (shelf.n.01, be.v.01, wall.n.01), (support.n.01, be.v.01, wall.n.01), (time_clock.n.01, have.v.01, lock.n.01), (time_clock.n.01, have.v.01, point.n.09), (mailbox.n.01, have.v.01, numeral.n.01), (clock.n.01, be.v.01, time_clock.n.01), (prison_guard.n.01, be.v.01, time_clock.n.01), (pipe.n.01, be.v.01, time_clock.n.01), (clock.n.01, be.v.01, wall.n.01), (numeral.n.01, along.r.01, wall.n.01), (support.n.01, be.v.01, wall.n.01)}*

After collecting and formatting this raw data, we performed several preprocessing steps by removing uncommon object nodes and relationships, removing paragraphs with very small or nonexistent scene graphs, and transforming each object and relationship type to its 'synset' – a representation of all the object or relationship's synonyms. Each synset can be translated to a numerical index using a dictionary we aggregated. For both of our approaches, we arrange our training data into train/development/test segments using 90%, 7%, and 3% of the data, respectively.

4.2 Evaluation method

For our object prediction experiments, our evaluation metric was relatively straightforward. For predicted multisets, we use F1 score:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

In addition, we compute a raw accuracy, which is a percentage of how many nodes were correctly identified and an order accuracy, which is the percentage of the nodes identified of the right order (that is, how many nodes had the right count associated with them).

For our end-to-end experiments we simply use the $precision$ metric from the F1 score to evaluate the quality of our end-to-end scene graph generation models.

Table 2: Results

Encoder	Decoder/Predictor	F1 (%)	Accuracy (%)
2-Layer Bi-LSTM, $d = 128$	2-Layer Dense	27.8	24.62
3-Layer Bi-LSTM, $d = 256$	2-Layer Dense	34.1	32.21
2-Layer Bi-LSTM, $d = 128$	Multitask Conv (3x Maxpool)	29.6	21.11
2-Layer Bi-LSTM, $d = 128$	Multitask Conv (2x Maxpool)	31.3	21.84
3-Layer Bi-LSTM, $d = 256$	3-Layer Dense	33.4	31.04
3-Layer Bi-LSTM, $d = 256$	Multitask Conv (2x Maxpool)	34.3	33.17
Dependency Parse+Align	2 Layer Dense $d = 128$ (object)	–	0.059
Dep. Parse+Align+BiLSTM $d = 128$	2 Layer Dense $d = 128$ (edge)	–	0.061

4.3 Experimental details

Outside of the dependency parser, our entire infrastructure was built from scratch, with each module of our model created in PyTorch, and the various training loops for each part of our model tailored to load our data in a way that was able to be input to our model. For each of our experiments, we used an Adam optimizer with the same default intializations as in previous assignments.

For our one-shot multi-set prediction experiments, we split the 18,550 data pairs into 16,695 training examples, 1,555 development examples, and 300 test examples. We trained each model to completion over 40 epochs over our dataset, evaluating on the development set ever 500 iterations. We also used a batch size of 128 examples in training. Table 2 shows our various model training configurations for each experiment run.

For our end-to-end experiments, we attempted various methods with which to train our models. First, we attempted to pre-compute our dependency and alignment algorithm outputs, so that these could be input into the object decoder in a batched manner. However, we found that this method created far too high of a RAM overhead to be tractable using our resources. Next, we created a modified training loop to batch small sets of dependency parses and alignment outputs, and then feed them into our object and edge models. However, due to a possible memory leak in our implementation, we were unable to get our model to run for more than one epoch without crashing our VM and, as such, report incredibly disappointing results. In our alignment step, we chose to not consider sentences which did not have any aligned ground truth relationships, and chose to cut off our Levenstein distance search at an edit distance of three. For our edge feature encoder, we chose a hidden size for the LSTM of 128. Finally, we did not fully implement the iterative message passing step as discussed previously, instead focusing on trying to fully train baseline end-to-end models which did not include iterative message passing.

4.4 Results

We report the F1 Node Prediction Scores and Node Prediction Accuracy Percentages for our first approach in Table 2. Adding additional complexity to our node prediction model in the form of multitask learning and convolutional blocks did not necessarily increase the performance of our system. In addition, an overall accuracy averaging from 25-30% and a max F1 score of 34.3 is not very accurate.

We were unable to successfully implement, train and test our designed end-to-end systems. This was due to a memory leak either in our alignment or dependency parsing system, which caused our (56 GB) VM to crash near the end of the first epoch of training. As a result, we report validation results at the latest point we were able to reach in training our end-to-end system. The models that we were able to train obviously failed to learn their respective tasks to any meaningful level, with precision metrics in the sub-percent range.

5 Analysis

In this section, we present several examples of model output for the various models we trained for the purpose of qualitative analysis. Since we used several different approaches, and built fully custom models for each approach, the outputs are formatted differently for each model. First, we analyze two (random) sample outputs of our one-shot multitask node prediction model:

- **Input Paragraph:** *A woman is standing in front of an elephant cage. An elephant is extending its trunk to the womans hand, which she is holding out to him. The woman has brown hair pulled back in a ponytail, and sunglasses on the top of her head. She is wearing a red skirt and tan tank top, and sandals on her feet. She has a brown purse strung over her shoulder.*

  Predicted Object SynSets: *[shoe.n.01, trunk.n.01, girl.n.01, airplane.n.01, shirt.n.01, elephant.n.01, elephant.n.01, elephant.n.01, elephant.n.01, elephant.n.01, man.n.01, window.n.01, sign.n.02, bench.n.01, clock.n.01, sheep.n.01, short_pants.n.01, person.n.01, person.n.01, person.n.01, person.n.01, ear.n.01, woman.n.01]*

- **Input Paragraph:** *This is an image of a sporting event. The woman is playing tennis. The woman is holding a tennis ball. The ball is light green. The woman is about to serve the ball The girl is holding a tennis racket. The girl is wearing a shirt. The shirt is white. The shirt has a design of a bird on it. The girl has on black shorts. The racket is orange and black.*

  Predicted Object SynSets: *[shoe.n.01, line.n.01, ball.n.01, shirt.n.01, court.n.01, short_pants.n.01, leg.n.01, woman.n.01, racket.n.04]*

These outputs gives us some insight into the performance of our multitask object prediction model. This model is obviously able to capture the general semantics of a scene. For example, when it is input a paragraph describing a tennis sporting event, it predicts 'line' such as the lines found on the court. This is done without a reference to 'line' in the source paragraph. Therefore, we can be confident that our encoding architecture is learning the general semantics of the paragraph in question to some degree. In addition, it is reasonably predicting objects that are present in the source paragraph. However, this model seems to have problems with predicting how many of each object is present in each input paragraph. This is likely because the model fails to accurately learn the grammatical structure of the paragraphs.

Now, we qualitatively analyze several non-cherry-picked outputs from our dependency parsing and alignment pipeline:

- **Sentence:** *There is a person sitting at the table.*

  Output: *Candidate Objects: [person, table], Alignments: [person:person.n.01, table:table.n.01], Relationships: [(person,have.v.01,table)]*

- **Sentence:** *The other elephants are in the forest.*

  Output: *Candidate Objects: [elephants, forest], Alignments: [elephants:elephant.n.01, forest:land.n.01], Relationships: [(elephant,along.v.01,forest),(forest,behind.v.01,elephants)]*

Here, we see that our dependency parsing and alignment systems are not completely failing to capture relevant information in the downstream scene graph. We did, however, see errors in aligning raw candidate objects to the target synsets to create viable training pairs. This is inherent to the unsupervised nature of our alignment algorithm, but we found this (alignment in general) to be a major bottleneck issue in training downstream models.

While we were not able to test the downstream node and edge prediction model due to implementation bugs and memory issues, results from [2] cause us to believe that our model design had a high likelihood of working if we had been able to successfully implement it in the time provided for this project.

6 Conclusion

We present novel methods and experiments that test these methods for the purpose of generating semantically rich scene graphs from paragraph level descriptions of images. Our two main approaches utilized a variety of techniques to predict node multisets and to predict object and relationship pairs present in natural language. While our models failed to precisely generate accurate scene graphs, and in some cases failed to train entirely due to computation restrictions, we see this as a first step in building a fully neural end-to-end system for generating scene graphs from natural language. This is a novel research question and, given more time, we would like to explore future work on using one-shot recurrent models and attention mechanisms, along with cleaner data.

References

[1] Schuster, Sebastian, et al. "Generating semantically precise scene graphs from textual descriptions for improved image retrieval." Proceedings of the fourth workshop on vision and language. 2015.

[2] Xu, Danfei, et al. "Scene graph generation by iterative message passing." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Vol. 2. 2017.

[3] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

[4] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[5] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).

[6] Krishna, Ranjay, et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123.1 (2017): 32-73.

[7] De Marneffe, Marie-Catherine, et al. "Universal Stanford dependencies: A cross-linguistic typology." LREC. Vol. 14. 2014.

[8] Johnson, Justin, et al. "Image retrieval using scene graphs." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[9] Chen, Danqi, and Christopher Manning. "A fast and accurate dependency parser using neural networks." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[10] Johnson, Justin, Agrim Gupta, and Li Fei-Fei. "Image generation from scene graphs." arXiv preprint (2018).

[11] You, Jiaxuan, et al. "Graphrnn: Generating realistic graphs with deep auto-regressive models." International Conference on Machine Learning. 2018.

[12] Haigh, Alex, Schwager, Sam, and van Paasschen, Frits. "Deep Autoregressive Models for Conditional Graph Generation". CS236 Final Project. 2018.

[13] Liu, Ying, et al. "A survey of content-based image retrieval with high-level semantics." Pattern recognition 40.1 (2007): 262-282.

[14] Jerry R Hobbs. 1978. Resolving pronoun references. Lingua, 44(4):311–338.

[15] Rezatofighi, S. Hamid, et al. "Deepsetnet: Predicting sets with deep neural networks." 2017 IEEE International Conference on Computer Vision (ICCV). IEEE, 2017.

[16] Vinyals, Oriol, Samy Bengio, and Manjunath Kudlur. "Order matters: Sequence to sequence for sets." arXiv preprint arXiv:1511.06391 (2015).

[17] Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. "Highway networks." arXiv preprint arXiv:1505.00387 (2015).

[18] Yang, Jianwei, et al. "Graph r-cnn for scene graph generation." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

[19] Krause, Jonathan, et al. "A hierarchical approach for generating descriptive image paragraphs." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[20] Collobert, Ronan, and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[21] Loper, Edward, and Steven Bird. "NLTK: the natural language toolkit." arXiv preprint cs/0205028 (2002).

[22] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[23] Lee, Kenton, Luheng He, and Luke Zettlemoyer. "Higher-order coreference resolution with coarse-to-fine inference." arXiv preprint arXiv:1804.05392 (2018).

[24] Dozat, Timothy, and Christopher D. Manning. "Deep biaffine attention for neural dependency parsing." arXiv preprint arXiv:1611.01734 (2016).

[25] De Marneffe, Marie-Catherine, and Christopher D. Manning. Stanford typed dependencies manual. Technical report, Stanford University, 2008.

[26] De Marneffe, Marie-Catherine, and Christopher D. Manning. "The Stanford typed dependencies representation." Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation. Association for Computational Linguistics, 2008.

[27] McCann, Bryan, et al. "The natural language decathlon: Multitask learning as question answering." arXiv preprint arXiv:1806.08730 (2018).

[28] Tamburini, Fabio. "Semgrex-Plus: a tool for automatic dependency-graph rewriting." Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017). 2017.

[29] Gardner, Matt, et al. "AllenNLP: A deep semantic natural language processing platform." arXiv preprint arXiv:1803.07640 (2018).

[30] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).